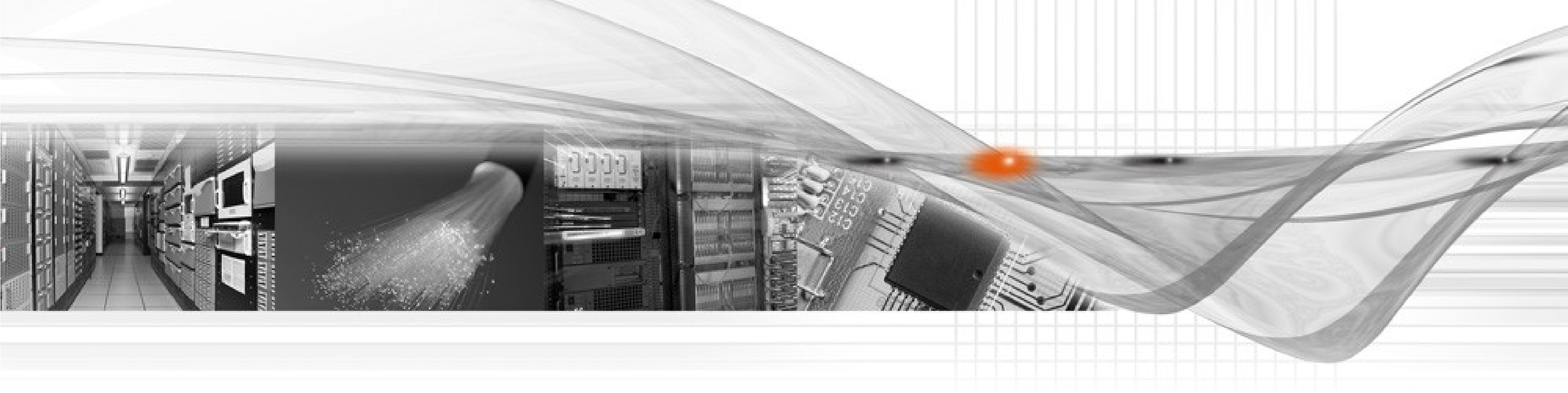


User Mode Ethernet Programming

OFA 2012

Author: Tzahi Oved

Date: March 2012

User Mode Ethernet – Why?

- Dramatically reduce operating systems overhead
- Improve network performance utilizing NICs support for user mode send/receive rings
  - High PPS rates, low latency, low CPU utilization and increased scalability
- Transparently use standard TCP/UDP/IP protocols
  - No need for proprietary protocol designs
  - Use existing rich HW protocol offload support
  - Can interoperate with traditional OS TCP/IP stack

User Mode Ethernet – How?

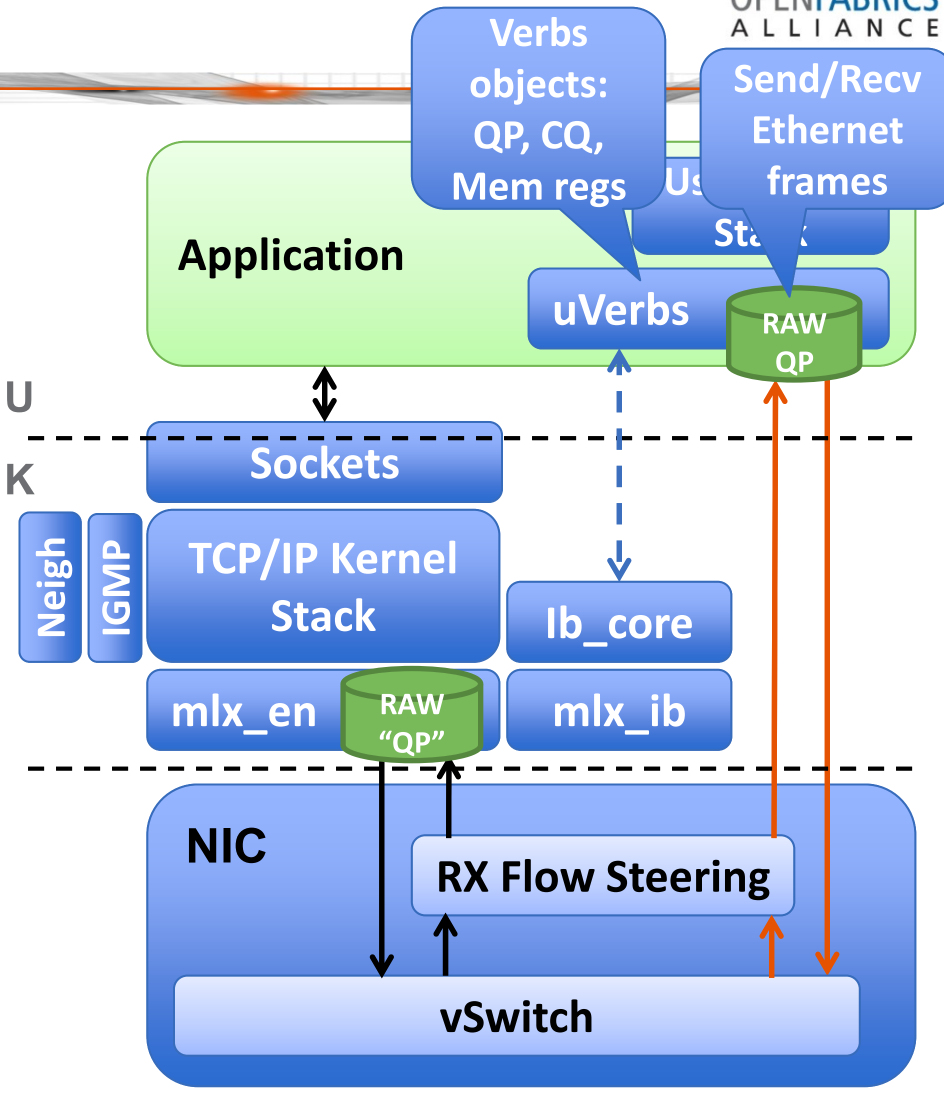
- Application needs:
  - A direct HW Send Queue that send raw packets
  - A direct Receive Queue that steers incoming flows
  - No headers are generated implicitly (only explicitly)
  - RX, TX completion queue

mmm… what can fit such requirements?

Hey! We’ve got Verbs

Raw QP

- QP Send queue to use raw packets
- QP receive queue is steered according to flows
- Reuse the mature stack of verbs: QP, CQ, mem registrations ops



QP as a User Mode Interface

- Receive flow based steering
- Port bonding and user mode LAG for HA & FO
- QoS
- Internal adapter switching
  - For intra-node/loopback communication
  - Built into adapter capabilities already
- Stateless offloads
- RSC and TSC
- Time stamping
- Completion interrupt moderation

Verbs Extensions

- The requirement
  - Make verbs extendable without breaking the ABI
  - Don’t make the API even more complicated
- ~~Option~~ 1
  - Create new verb call: ibv_xxx_ex() for new operations
  - Tomorrow, create ibv_xxx_ex2() for newer
- Option 2
  - Use ABI Versioning
  - ibv_xxx_ex() will always check ABI ID
  - Verb call input parameters struct size is defined by ABI ID
  - ABI ID can be set by ibv_open_device() or handed in each xxx_ex() call
  - ABI ID to include general version and vendor specific info

User Mode Stateless Offloads

User Mode Stateless Offloads

- Capabilities
  - Checksum offload
  - Ethernet L2 header stripping and insertion
    - Subset is VLAN (VID + Priority) striping and insertion
  - Large Send Offload for TCP: HW segmentation
  - Large Receive Offload for TCP: HW de-segmentation
- Verbs API
  - ibv_query_device() to expose offload caps through ibv_device_attr->device_cap_flags
  - For QP type = RAW (and UD) extend verbs create_qp:
    - ibv_create_qp(ibv_pd*, ibv_qp_init_attr), where:
    - ibv_qp_init_attr to include ibv_create_qp_flags
  - ibv_create_qp_flags:
    - L3, L4 TX and RX checksum create/verify offload
    - VLAN: VID+Priority, whole L2* strip/insert
    - Enable flow steering
    - LSO, LRO*

***Not supported in current HW**

User Mode Stateless Offloads

- Verbs API
  - For VLAN and ETH* strip:
    - ibv_wc to include: d.MAC, s.MAC, VLAN
  - For VLAN, ETH insertion and LSO:
    - ibv_send_wr to include new struct in wr union for RAW QP
    - wr union
      - struct rdma
      - struct atomic
      - struct ud
      - struct raw // New
        - » ibv_ah // Points to d.MAC, s.MAC, VLAN (VID+prio)
        - » mss (for LSO, relevant if LSO is enabled)
  - For RX csum
    - ibv_wc to include: L3, L4 csum verification result
    - Relevant when any rx csum is enabled

*Not supported in current HW

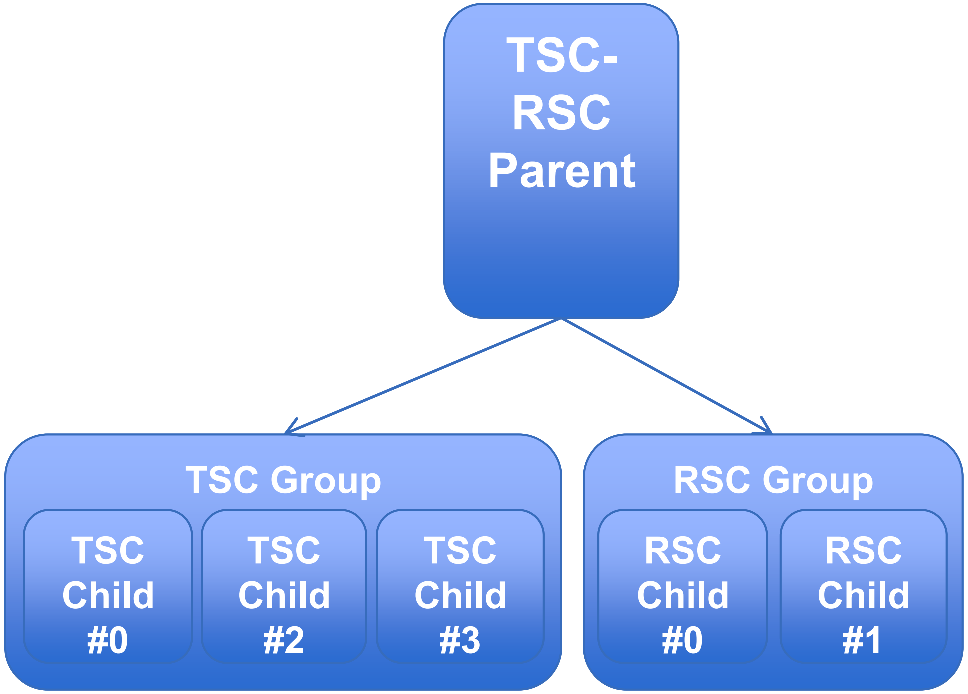
User Mode Receive and Transmit Scaling

Receive and Transmit Scaling

- Use multiple receive and send rings for IO operation
- Scale network handling with growing core count
- Apply NUMA locality of context and I/O operation
  - Receive buffer is close (mem access wise) to receiving context

TSC/RSC Model

- TSC/RSC parent
  - Comprises TSC and RSC QP groups
- TSC group
  - A group of QPs with same source address
- RSC group
  - A group of QPs that are the target of RSC hashing
- All QPs are manipulated through regular Verbs
- Same model applies to IPoIB as well



Verbs API

- Use ibv_create_qp() and ibv_destroy_qp() to create/destroy parent and children QPs
- struct ibv_qp_init_attr to include:
  - enum ibv_qpg_type // QP Group type
    - IBV_QPG_NONE // Not a QP Group type
    - IBV_QPG_PARENT // RSC and/or TSC Parent QP
    - IBV_QPG_CHILD_RX // RSC child QP
    - IBV_QPG_CHILD_TX // TSC child QP
  - struct ibv_qpg_init_attrib // Valid for ibv_qpg_type= IBV_QPG_PARENT, includes:
    - TSC child count (TSC Vector size)
    - RSC child count (RSC Vector size)
    - struct ibv_qp ibv_qpg_parent: points to parent QP
      // Valid for Ibv_qpg_type = IBV_QPG_CHILD_RX or IBV_QPG_CHILD_TX
- Only ibv_qp_type = RAW or UD are supported

Verbs API – cont.

- Enable QP Group **parent** characteristics modification
- **struct** ibv_qp_attr to include:
  - struct ibv_qpg_attrib for QP Group type attributes, includes the following members:
    - RSC Hash func type
    - RSC Hash header fields selection: IPv4, IPv6, UDP, TCP
    - RSC Key
    - RSC indirection table update
- Where ibv_qp_attr_mask to indicate updated type

Time Stamping Completion Interrupt Moderation

User mode Time Stamping

- Enable accurate user mode statistics and tracking
- Perform time stamp on transmit and receive events
- Expose OS bypass time stamping facilities
- Expose adapter HW clock directly to user mode
- Similar Verbs API can be used from kernel as well
  - Kernel ULPs can use it too
- Can be used for:
  - Application latency measurement
  - Packet tracking/logging
  - Other..

Verbs API

- Expose per CQ interrupt moderation
- Expose CQ time stamping for receive and transmit completions
- Add (exists in kernel, but requires extension):
  - ibv_modify_cq (ibv_cq*, ibv_cq_attr*, int attr_mask)
- ibv_cq_attr to include:
  - enum ibv_ts_type // For Time stamping enablement and type
- Where enum ibv_ts_type:
  - TS_TYPE_NONE // No TS for the CQ please
  - TS_TYPE_RAW // Provide raw adapter free running clock
  - TS_TYPE_TOD // Provide translated TS to Time Of Day
- ibv_poll_cq(ibv_cq*, num_entries, ibv_wc*)
  - Where struct ibv_wc to include u64 time stamp value
- ibv_query_cq() // May be added to provide CQ characteristics

User Mode Interrupt Moderation

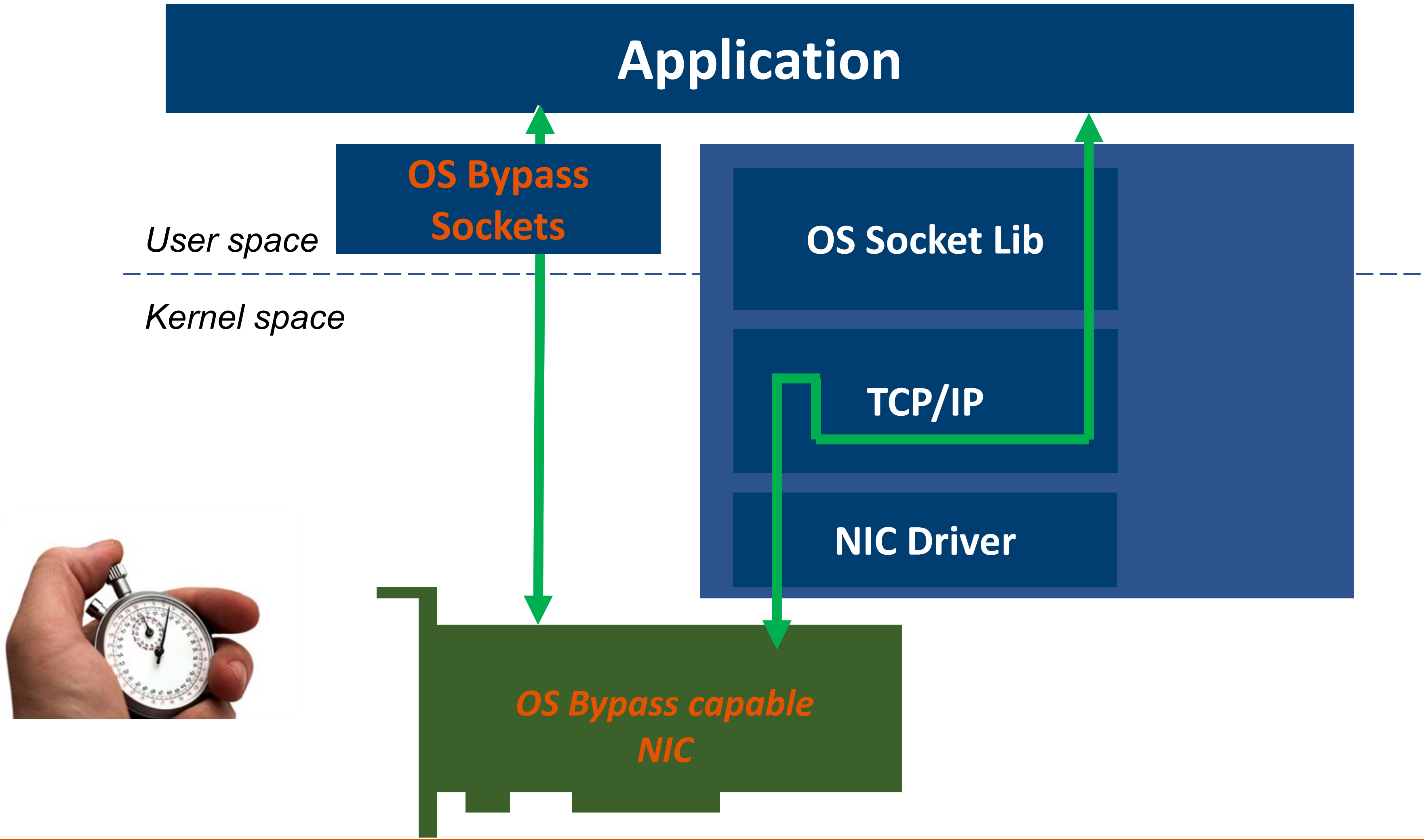
User Mode Interrupt Moderation

- Today supported from kernel only
- Extend to user mode
- Verbs API
  - ibv_modify_cq (ibv_cq*, ibv_cq_attr*, int attr_mask)
  - Where ibv_cq_attr to include:
    - count   // number of CQEs to trigger an event
    - period   // max period in usec prior to triggering an event
- ibv_query_cq() // May be added to provide CQ characteristics

OS Bypass Sockets

Complete accelerated user mode stack

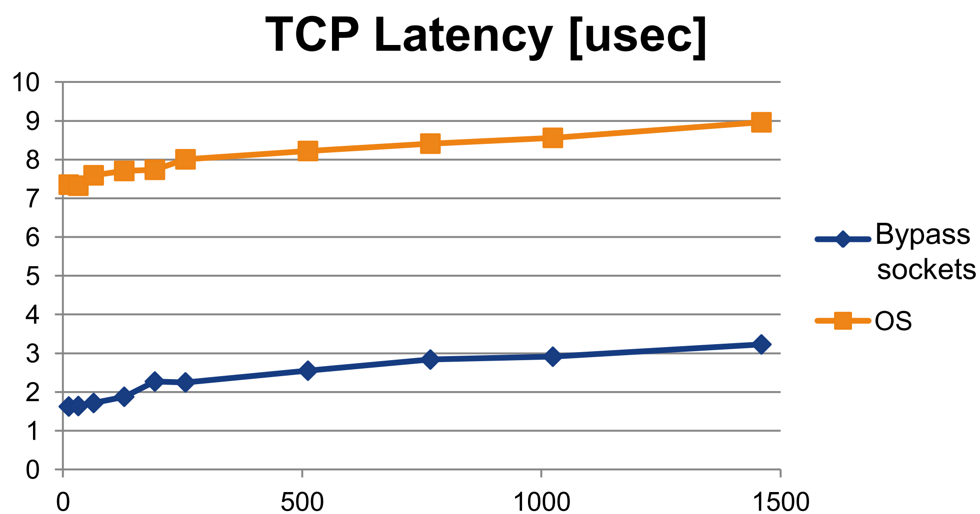
Complete OS Bypass



Performance Numbers – TCP

	Bypass				OS	
MsgSize [Bytes]	MsgRate [mps]	BW [Gbps]		MsgSize [Bytes]	MsgRate [mps]	BW [Gbps]
12	4,676,836	428		12	636,043	58
32	5,577,214	1,362		32	723,038	177
64	5,194,631	2,536		64	691,814	338
128	4,657,394	4,548		128	661,711	646
192	4,464,682	6,540		192	606,785	889
256	3,851,569	7,523		256	718,902	1,404
512	2,742,317	10,712		512	642,647	2,510
768	2,109,650	12,361		768	609,287	3,570
1024	1,780,715	13,912		1024	677,960	5,297
1460	1,497,094	16,676		1460	642,576	7,158

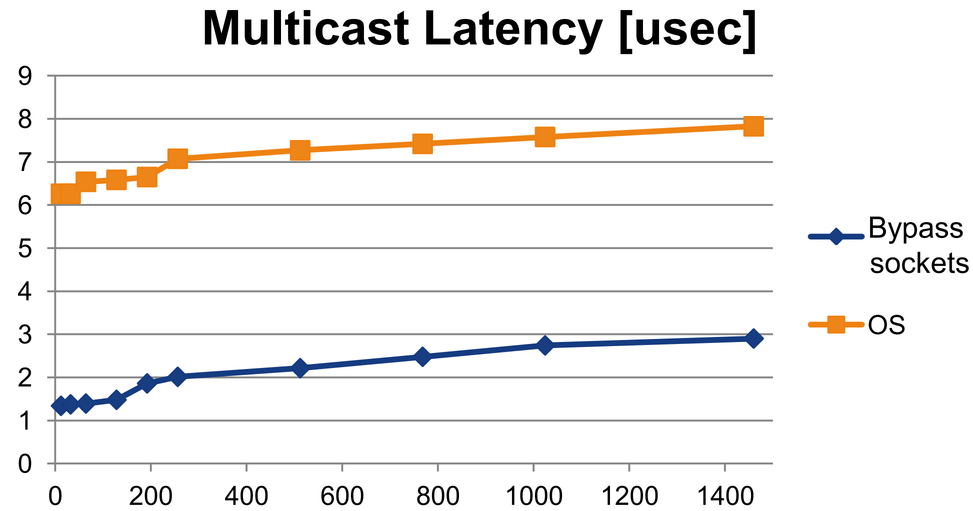
Msg Size [Bytes]	Latency [usec]	
	Bypass	OS
12	1.625	7.352
32	1.638	7.325
64	1.716	7.593
128	1.876	7.706
192	2.270	7.736
256	2.248	8.006
512	2.549	8.221
768	2.837	8.411
1024	2.911	8.560
1460	3.228	8.961



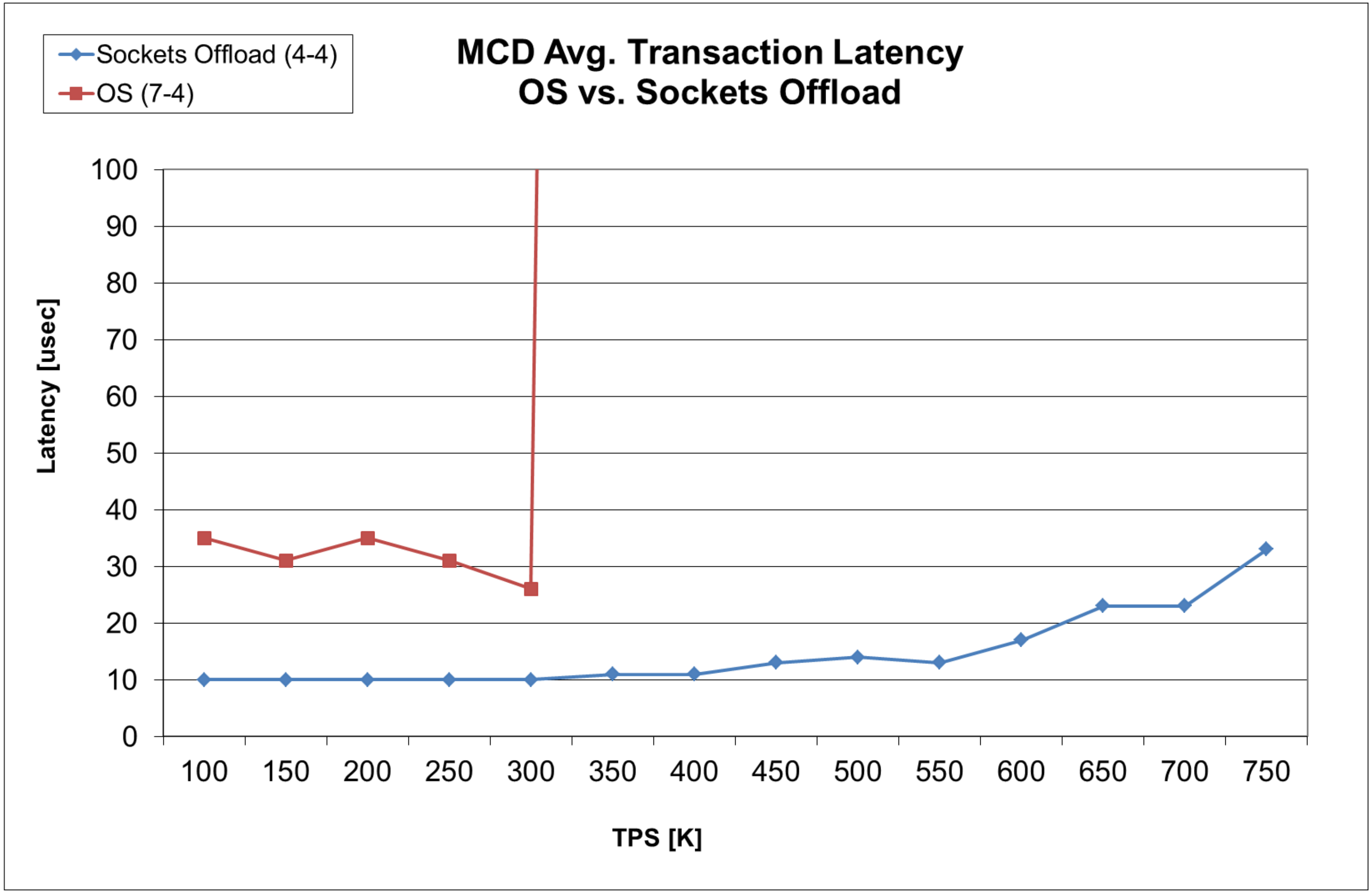
Performance Numbers – Multicast

	Bypass				OS	
MsgSize [Bytes]	MsgRate [mps]	BW [Gbps]		MsgSize [Bytes]	MsgRate [mps]	BW [Gbps]
12	3,583,267	328		12	1,121,512	103
32	3,581,469	874		32	1,060,979	259
64	3,247,848	1,586		64	1,081,166	528
128	3,064,820	2,993		128	1,080,604	1,055
192	3,114,609	4,562		192	1,075,363	1,575
256	3,136,012	6,125		256	1,075,704	2,101
512	2,151,377	8,404		512	1,059,383	4,138
768	2,155,998	12,633		768	983,353	5,762
1024	1,911,766	14,936		1024	973,862	7,608
1460	1,741,187	19,395		1460	987,417	10,999

Msg Size [Bytes]	Latency [usec]	
	Bypass	OS
12	1.339	6.262
32	1.373	6.263
64	1.392	6.538
128	1.479	6.583
192	1.859	6.649
256	2.012	7.073
512	2.213	7.269
768	2.477	7.419
1024	2.741	7.579
1460	2.901	7.826

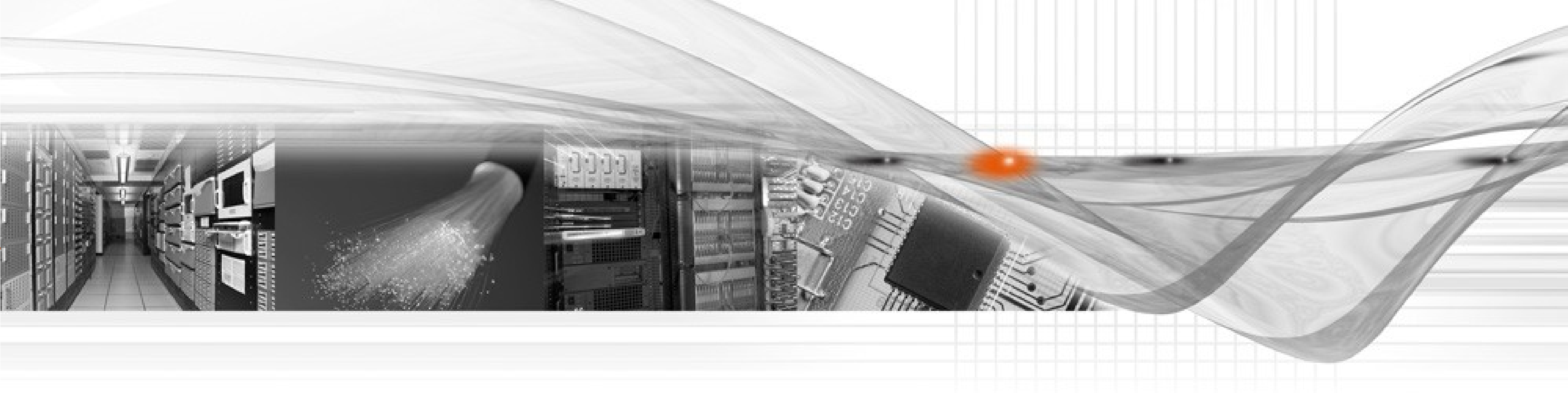


Application example: memcached



Summary

- User mode Ethernet is well suited for verbs API
- Reuse existing mature elements of verbs
- Capable of providing NIC standard and new offloads
- But now each flow can have it dedicated “NIC”
- Ready for your next user mode stack



OPENFABRICS ALLIANCE

Thank You